How speedy is SPDY?

Kumar Suyash (113277210)

For a decade, the web infrastructure has been based upon HTTP/1.1. However, HTTP/1.1 has its own drawbacks - like not providing compression for the headers and allowing multiple connections with the same server for a client - leading to wastage of resources. In order to overcome this, Google came up with SPDY.
The paper is a detailed study of SPDY - first of its kind, by comparing the page load time (PLT) of HTTP/1.1 and SPDY - experimented over top 200 Alexa sites. SPDY differs mainly from HTTP/1.1 in terms of the number of TCP connections allowed from a client to a server - and the fact that SPDY tries to use pipelining to it's advantage. Upon the completion of these experiments - we try to evaluate if SPDY is actually faster than HTTP/1.1 and if yes, then under which conditions.
Running such a comparison is, however, challenging since both the protocols' performance depends on many external factors and thus to include all these factors, we need to consider a large number of parameters. Another challenge is of the PLT's variability - stemming from factors like browser computation.
The paper claims that upon negating the browser dependencies and computation - SPDY improves PLT for small object sizes and with low loss rates - since it batches these small objects into a TCP segment - reducing the retransmission. The paper also finds that SPDY performs worse for high packet losses for large objects. Upon incorporating the dependencies though, the benefits of SPDY are marginalised. The paper further explores prioritization and server push - two characteristics of SPDY - and finds that although prioritization doesn't really help in improving the performance but server push does show potential.
SPDY overcomes the deficiencies of HTTP/1.1 by allowing just one TCP connection to a server - prioritizing pipelining over parallelization, a request prioritization for objects, allowing objects to be pushed from server side even without client explicitly asking for them and finally compression for the headers.
For running the experiments, the paper tries to untangle the various factors affecting the performance - so as to evaluate each measure separately. First the experiments are performed with ignoring page load dependencies and browser computation on both synthetic and real pages. For the experimental setup, the authors have developed their own SPDY client. This client instead of creating a new TCP connection per request, reuses these connections. A HTTP client is also developed for comparison where the maximum number of parallel connections are set to 6. To experiment with synthetic pages - objects with specific size and numbers are created. A broad range of parameters are considered for the same - like object size, RTT, etc. Now to understand comprehensively the effect of each parameter, a decision tree styled predictive model is considered where each configuration is a combination of these factors. For each configuration a variable s is added which is PLT of SPDY divided by PLT of HTTP. For $s < 0.9$ - SPDY outperforms HTTP and $s > 1.1$ - HTTP outperforms SPDY. The decision tree analysis generates the likelihood of a configuration working better under SPDY. This analysis suggests the SPDY performs worse when packet loss is high but well when there are small or large objects with low loss. It also suggests that object size and loss rate are the most important factors in predicting SPDY performance. To get a better understanding of these results the network traces are analyzed. The reason why SPDY works well on small objects is inferred to the fact that unlike HTTP/1.1, the SPDY batches the small objects and sends it across while with HTTP even sending a few small objects would use up the congestion window. Having a single TCP connection also helps with reducing the re-transmission loss and reducing the amount of time the pipe is idle. However, SPDY hurts with high loss since it reduces the congestion window much more than HTTP which reduces on only one of its parallel connections. In order to mitigate this loss, the paper suggests modifications in TCP - TCP+.
With TCP+ - the congestion window is automatically tuned according to a suggested formula. Upon performing the same set of experiments with TCP+ - it is seen that this new TCP configuration performs better than the original TCP one. Thus, by tuning the congestion window - the negative performance of SPDY under high loss can be mitigated. For real web pages which incorporate dependencies and computation - Epload is created. It can separate the network operations

and computations and is used to measure the impact of dependencies and computation. Upon incorporating these dependencies it is seen that the effect of SPDY is largely decreased.

Upon exploring prioritization and server push - it is seen the impact of explicit prioritization is negligible and server push does help especially under high RTT. Upon incorporating the TCP+ changes and server push changes suggested by the author - SPDY's performance increases by about 30% under high RTTs.

The paper however has limitations of it's own. The paper has not performed a deep study on mobiles websites - which today account for a significant portion of network traffic. The server load and processing is also not taken into account.

However, the paper did introduce me to the contributing factors to SPDY's performance other than browser computation - factors like RTT, packet loss, object size, etc. I also learnt about how decision trees can be used to analyze various configurations when faced with a number of factors.

I can use these decision tree findings to create various configurations for my project on real estate price estimations when provided with a large number of factors like crime rates in a locality, number of rooms in a house, etc.